Attorney's Docket No.: 13913-166001/2003P00111US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: Johannes Lauterbach, et al.	Art Unit	: Unknown
Serial No.	: 10/789,949	Examiner	: Unknown
Filed	: February 27, 2004		
Title	: PROVIDING RUNTIME OBJECT BY INSTANTIATING TEMPLATE-DERIVED CLASSES		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

TRANSMITTAL OF PRIORITY DOCUMENT UNDER 35 USC §119

Applicant hereby confirms his claim of priority under 35 USC §119 from the following application(s):

·European Patent Convention Application No. 03004489.5 filed February 28, 2003

·European Patent Convention Application No. 03012120.6 filed May 30, 2003

A certified copy of each application from which priority is claimed is submitted herewith.

Please apply any charges or credits to Deposit Account No. 06-1050.

CERTIFICATE OF MAILING BY FIRST CLASS MAIL

I hereby certify under 37 CFR §1.8(a) that this correspondence is being deposited with the United States Postal Service as first class mail with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

4/30/2004
Date of Deposit

Peggy C. Gray
Signature

Peggy C. Gray
Typed or Printed Name of Person Signing Certificate

Respectfully submitted,

Date: 4/30/04

Spencer C. Patterson
Reg. No. 43,849

PTO Customer No. 32864
Fish & Richardson P.C.
5000 Bank One Center
1717 Main Street
Dallas, Texas 75201
Telephone: (214) 292-4082
Facsimile: (214) 747-2091

90074735.doc

Europäisches Patentamt

European Patent Office

Office européen des brevets

Bescheinigung

Certificate

Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. **Patent application No.** **Demande de brevet n°**

03004489.5

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

Anmeldung Nr:
Application no.: 03004489.5
Demande no:

Anmeldetag:
Date of filing: 28.02.03
Date de dépôt:

Anmelder/Applicant(s)/Demandeur(s):

SAP Aktiengesellschaft
Neurottstrasse 16
69190 Walldorf
ALLEMAGNE

Bezeichnung der Erfindung/Title of the invention/Titre de l'invention:
(Falls die Bezeichnung der Erfindung nicht angegeben ist, siehe Beschreibung.
If no title is shown please refer to the description.
Si aucun titre n'est indiqué se referer à la description.)

Processing development object into runtime object

In Anspruch genommene Prioriät(en) / Priority(ies) claimed /Priorité(s) revendiquée(s)
Staat/Tag/Aktenzeichen/State/Date/File no./Pays/Date/Numéro de dépôt:

Internationale Patentklassifikation/International Patent Classification/
Classification internationale des brevets:

G06F17/60

Am Anmeldetag benannte Vertragstaaten/Contracting states designated at date of filing/Etats contractants désignées lors du dépôt:

AT BE BG CH CY CZ DE DK EE ES FI FR GB GR HU IE IT LU MC NL
PT SE SI SK TR LI

PROCESSING DEVELOPMENT OBJECT INTO RUNTIME OBJECT

Field of the Invention

01 The present invention generally relates to data processing and, more particularly, relates to computer systems, computer programs, and methods to provide a runtime object with computer code to control business applications.

02 Background of the Invention

03 Software manufacturers design business and enterprise applications at design time, business organizations ("customers") use the business applications at run time.

04 Software manufacturers concentrate on the business requirements of their customers and provide applications that are customized for various platforms and requirements.

05 The applications are implemented with application specific code (hereinafter "code") in computer languages such as C++, Java or Visual Basic for Applications (VBA). Using object-oriented languages is convenient so that development objects are processed to runtime objects.

06 One manufacturer usually serves multiple customers. The runtime objects need to be adapted to the particular needs of a particular customer. Therefore, the manufacturer provides an interpreter in combination with a plurality of templates. The customer uses the interpreter to provide the development objects by interpreting the templates in view of customer-specific data.

07 The applications are implemented on specific runtime platforms, or frameworks. Different platforms may require runtime objects in different languages to adapt to different operating systems.

08 There are several disadvantages of the prior art. The interpreter is complex and specifically adapted to the runtime framework. The manufacturer needs to send the code template and the interpreter to the customer.

09 Type consistency between development objects needs consideration, especially when a development object or the template is modified. Accidental and intentional (even malicious) changes to the templates (especially at the customer site) could lead to inconsistencies in the code.

010 There is an ongoing need to provide improved method, systems, an computer programs to provide runtime objects (source code).

011 Summary of the Invention

012 According to the present invention, a method for use in a computer relates to processing a development object (DO) into a runtime object. The method comprises transforming a development object (DO) into an intermediate object (IO), building an abstract syntax tree (AST) from the intermediate object by using a template, and generating the runtime object from the abstract syntax tree while preserving the structure of the template.

013 It is advantageous that different runtime objects for different runtime frameworks can be provided based one and the same intermediate object (IO). In other words, the intermediate object becomes the standard for deriving runtime objects for different runtime frameworks. Preferably, the intermediate object does not contain specifics of the runtime framework. It is further advantageous that the method can be performed at a single functional entity at a single time point without interference from other functional entities.

014 Further, the runtime code - being the target - is no longer part of the code interpreter. Instead, the code is part of the templates. Mistakes or errors are more difficult to make.

015 The portability between languages is enhanced. Switching between languages with same semantic but different syntax, it is sufficient to change the templates. Changing the runtime code can be accomplished by modifying the templates or the development object.

016 The above-mentioned problem is solved by method, system and computer program according to the independent claims; preferred implementations are stated in the dependent claims.

017 <u>Brief Description of the Drawings</u>

018 FIG. 1 illustrates an exemplary computer architecture for implementing the present invention, wherein the computers are operated according to a developer (DEV) function, a processing (PRO) function and a run (RUN) function;

019 FIG. 2 illustrates an overview about development objects and runtime objects;

020 FIG. 3 illustrates an exemplary overview about runtime objects that can be provided according to the present invention;

021 FIG. 4 illustrates a simplified flow chart diagram of a method of the present invention;

022 FIG. 5 illustrates a diagram with software components to implement the method of the present invention;

023 FIG. 6 illustrates details for step transforming;

024 FIG. 7 illustrates further details for step transforming by comparing a first model of the development object and a second model of an intermediate object;

025 FIG. 8 illustrates details for step building;

026 FIG. 9 illustrate details for step building by showing the abstract syntax tree as a diagram in UML for an exemplary user interface;

027 FIG. 10 illustrates further details for step building by showing the names and properties for each node in the diagram of FIG. 9;

028 FIG. 11 illustrates details for step building by showing the abstract syntax tree of FIGS. 9-10 as a file with code segments for each class/node;

029 FIG. 12 illustrates a classification of templates and corresponding abstract syntax trees, by examples from "application class" to "business object class";

030 FIG. 13 illustrates a division of templates into template files by example for the "application class" template, with the assumption that one template file has one production rule;

031 FIG. 14 illustrates an example template file with an example production rule "BCDeclaration";

032 FIG. 15 illustrates providing an abstract syntax tree library;

033 FIG. 16 illustrates an example of a generator, with computer instructions perform step generating (of the method of FIG. 4) from the abstract syntax tree (AST), thereby instantiating the AST with data;

034 FIG. 17 illustrates exemplary data, used during step generating;

035 FIG. 18 illustrates an example for an application class file being a runtime object, obtained in step generating;

036 FIG. 19 illustrates an example for a generation template, used to provide a generator (for step generating, cf. FIG. 16);

037 FIG. 20 illustrates an example of an AST-XML-template (result of converting generation template to AST-XML-template by using template-to-XML-converter) used for providing the generator;

038 FIG. 21 illustrates an example of an XSL stylesheet used to provide the generator; and

039 FIG. 22 illustrates a simplified diagram of a computer network system that can be used to perform the method.

040 Detailed Description

041 The following description is presented to enable any person skilled in the art to make and use the invention, and is provided in the context of a particular application and its requirements. Various modifications to the disclosed implementations will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other implementations and applications without departing from the spirit and scope of the present invention. Thus, the present invention is not intended to be limited to the implementations shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.

042 Whenever possible, the same reference numbers and acronyms will be used throughout the figures to refer to the same or like elements. For convenience, reference lists are provided at the end of the specification.

043 An example for processing a development object (DO) into a runtime object will be presented for the user interface part of the application.

044 Conveniently, words are given in singular (e.g., development object, runtime object, computer, customer). The word "typical" (and its variations) refers to an implementation invention that is convenient but not mandatory. Details for an exemplary computer network system to use the method are explained at the end of the specification.

045 The following glossary introduces naming conventions.

046 The term "development objects" stands for any definition of the application, such as the behavior definition for user interface (UI) elements
For example, development objects have the form of flowcharts, models, model diagram.

047 The term "runtime object" stands for any set of computer instructions that can be invoked to run on a computer to perform the application or parts of the application (e.g. the user interface).

048 The term "element" stands for information components (of a document) such as sections, lists, or paragraphs.

049 The term "abstract syntax tree" (AST) stands for any computer-intemal representation of a runtime object. The AST can be illustrated by a diagram (i.e. tree with nodes, cf. FIGS. 9-10) and that can be coded, for example, by a plurality of code lines (cf. FIG. 11).

050 The term "production rule" stands for a predefined section of text used for providing a code class in the runtime object and corresponds to a node in the abstract syntax tree.

051 In the figures, rectangles with round comers stand for computer instructions that are processed; rectangles with sharp comers stand for computer instructions that cause processing.

052 FIG. 1 illustrates an exemplary computer architecture for implementing the present invention. The figure concentrates on functional and time aspects of the invention.

053 For convenience of explanation, computer operation is classified into 3 functions. A typically operation scenario includes that these 3 functions are performed on 3 different computers at 3 different time periods, respectively.

054 The functions are developer (DEV) function 1000; processing (PRO) function 2000 (also: "service engineer function" or consultant function), and run-time (RUN) function 3000 (also: "use function"). DEV 1000 is typically affiliated with the manufacturer; PRO 2000 and RUN 3000 are typically affiliated with the customer.

055 Typically, each function is performed by a person with a specialized skill set: by a developer (DEV 1000), by a consultant as processing specialist (PRO 2000), and by a user (RUN 3000). This is, however, not limiting, the functions can be performed by more or less persons, even by a single person.

056 Typical time periods are design time (prior to performing the method), process time (executing method), and run time (using method results). Conveniently, time process follows the TIME arrow from left to right. The arrow does not indicate the length of each period. Usually, the process period is the shortest period.

057 DEV 1000 takes care about development object (DO) 105 stored in repository 101, typically at design time. DEV typically 1000 operates application development environment 106. Typically, development objects 105 are object oriented (OO) objects.

058 PRO 2000 commands a computer to perform method 401. Typically, processing according to the invention starts when repository 101 and template 151 are coupled to any of transformer (T) 210, builder (B) 221 and generator (G) 241. T, G and R symbolize software components with instructions to perform method steps.

059 RUN 3000 stands for executing the application (in runtime framework 310 on the third computer) by using the runtime objects (RO) during run-time. Runtime objects 305 (RO) are files in in source code 301. RO 305 are the template-enhanced equivalents of the development objects 105 (DO).

060 It is an advantage (of the present invention) that if natural persons perform functions 1000, 2000 and 3000, they can use their skill sets for their particular functions without interfering into other functions. For example, the consultant does not need the skills of the developer and of the user.

061 Application development environment 106 stands for an environment to customize applications by interacting (read from / write to) with repository 101.

062 In view of the time of use (design time), repository 101 is also referred to as design time repository (DTR). Repository 101 stores settings that determine the behavior of an application (e.g., tab-order, popup usage).

063 Runtime framework 310 stands for a framework to process runtime objects 305 that are specific for the application.

064 FIG. 2 illustrates an overview about development objects 105 and runtime objects 205.

065

066 FIG. 3 illustrates an exemplary overview about runtime objects 305 that can be provided according to the present invention. Objects 305 are objects in source code 301.

067 FIG. 4 illustrates a simplified flow chart diagram of method 401 of the present invention

068 Method (401) for use in a computer for processing a development object (DO, 105) into a runtime object (305), the method (400) comprising the following steps: transforming (410) a development object (DO, 105) into an intermediate object (IO, 215); building (421) an abstract syntax tree (AST, 241) from the intermediate object (215) by using a template (151); and generating (431) the runtime object (205) from the abstract syntax tree (241) while preserving the structure of the template (151).

069 Preferably, method 401, wherein the development object (DO, 105) comprises meta-data for an application with information about the business logic of the application.

070 Preferably, method 401, wherein the development object (DO, 105) has been provided in a visual environment by drag and drop declarations.

071 Preferably method 401, wherein the runtime object (305) is an object in source code (301).

072 Preferably, method 401, wherein step transforming (410) involves a development object (DO) of a business application. By providing development objects (DO) for the business application, the software manufacturer concentrates on the business requirements of its customer. Providing runtime objects for the particular runtime framework (platform) in use by the customer is done automatically.

073 Preferably, method 401, wherein in step building (421), the template (151) uses language elements suitable for files selected from the group of: application class file, application project file, common registry file, machine specific registry file, business component class file, tileset class file, tile

HTML file, and business object class file.

074 Preferably, method 401, wherein generating (431) into source code (301) comprises to generate runtime objects in languages selected from the group of Java, Visual Basic for Applications (VBA), Hyper Text Markup Language (HTML).

075 Preferably method 401, wherein generating (431) the runtime object comprises to replace placeholders in the abstract syntax tree (241) with data. Elements in the runtime objects (e.g., source code) that are independent from the structure of the development object are completed upon generating the runtime objects. For example, the element "calendar date indicating completion of source code" is introduced by the placeholder "&GenDate&"). Such data is independent from the software manufacturer.

076 Preferably, method 401, wherein transforming (410) comprises to receive the development object (105) from a repository (101). The repository is a database that stores development objects and that manages versions of the objects. The repository exists during design time and during transition time. Preferably, the repository is read-only.

077 Preferably, method 401, wherein transforming (410) is performed for a plurality of development objects (105).

078 Preferably, method 401, wherein transforming (410) the plurality of development objects (105) comprises to preserve the relations between the development objects (105).

079 Preferably, method 401, wherein transforming (410) comprises to use development objects (105) based on a first model and to provide intermediate objects (215) based on a second model, wherein the first

model and the second model have the same meta-model.

080 Preferably, method 401, wherein transforming (410) comprises to keep the properties and relations in the first model and in the second model.
In other words, the first model that is underlaying the development objects (105) is converted to the second model of intermediate objects (215) (i.e. intermediate object model). The underlaying structures of both models (i.e. their meta-model) remains the same.

081

082 Preferably, method 401, wherein transforming (410) comprises to use development objects (105) based on a first model and to provide intermediate objects (215) based on a second model, wherein the first model and the second model have a different meta-model.

083 Preferably, method 401, wherein transforming (410) comprises to use development objects (105) based on a first model and to provide intermediate objects (215) based on a second model, wherein the meta-model for the first model is a subset of the meta-model of the second model.

084 Preferably, method 401, wherein transforming (410) comprises to convert a definition of the first model from a first form in UML (274) to a second form in XML (275) and to add particulars of the second model by XSL (277).

085 Preferably, method 401, wherein transforming (410) comprises to convert from the definition in XML to the intermediate objects by using XSLT (276).

086 Preferably, method 401, wherein transforming (410) comprises to merge the definition in XML with an XSL stylesheet by an XSLT processor.

087 Preferably, method 401, wherein transforming (410) comprises to realign relations between development objects (105).

088 Preferably, method 401, wherein in step building (421), the syntax tree (241) is provided in a language that has common elements to the source code language.

089 Preferably, method 401, wherein in step building (421), the syntax tree (241) is provided with primary entities (e.g., production rules) in each node of the tree.

090 Preferably, method 401, wherein in step building (421), the abstract syntax tree (241) is provided using an XSL-engine.

091 Preferably, method 401, wherein in step building (421), a template (151) is used that comprises elements selected from the group of:
property declarations, initialize statements, methods (the term "method" being used in the context of object oriented programming), event handlers, and layout definitions.

092 Preferably, method 401, wherein in step generating (431), the elements of the templates are converted into elements in the language of the run-time objects (305).

093 Preferably, method 401, step building (421) comprises
converting a generation template ( ) from a template grammar to an intermediate template ( ) in XML grammar, merging an XSL stylesheet to the abstract syntax tree (241) by an XSLT processor.

094 Preferably, method 401, wherein building (421) an abstract syntax tree (241) comprises to use multiple templates.

095 Preferably, method 401, wherein building (421) comprises to use multiple templates that are combined to a template project.

096 Preferably, method 401, wherein building (421) comprises to use multiple templates having production rules of different names within the template project.

097

098 Preferably, method 401, wherein building (421) comprises to use a template with an include statement.

099 Preferably, method 401, wherein building (421) comprises to add instructions to the abstract syntax tree that are adapted to control a computer to perform step generating (431).

0100 Preferably, method 401, wherein step building (421) comprises using a plurality of combined templates.

0101 Preferably, method 401, wherein in step building (421), the plurality of templates has names of production rules that are also names of the nodes in the abstract syntax tree.

0102 Preferably, method 401, wherein in step building, a template (151) is used that has a context free grammar. Such grammars are well know in the classification by Noam Chomski as grammar of type 2.

0103 Preferably, method 401, wherein step building is performed by processing the template with a stack machine.

0104 Preferably, method 401, wherein computer instructions to perform step generating are part of the template (151). In other words, instructions to perform generating, in short: "generating code" can be supplied to the computer together with the templates or as part of the templates. This alleviates the operator of the computer (e.g. the consultant) from using specific generating code.

0105 Preferably, method 401, wherein the computer instructions to perform step generating are in root of the template (151).

0106 Preferably, method 401, wherein step building (421) is repeated with a further template so that the abstract syntax tree if provided for a different language.

0107 Preferably, method 401, wherein step transforming (410) and building (421) both comprises using an XSLT processor.

0108 Preferably, method 401, wherein generating (431) comprises to replace placeholders in the bodies of production rules (i.e. of syntax tree nodes) by actual parameter values.

0109 Preferably, method 401, wherein generating (431) comprises to receive a parameter value and to replace a placeholder with that parameter value.

0110 Preferably, method 401, wherein generating (431) comprises to receive a parameter numeral and to replace a placeholder with a predefined value that corresponds to the parameter numeral. (An example is given in connection with "Enum").

0111 Preferably, method 401, wherein generating (431) comprises to provide one class of source code to one node of the abstract syntax tree.

0112 Preferably, method 401, wherein the steps building (421) and generating (431) are performed with such definitions (in the template) that the source code (301) that is executed in the runtime framework of a platform selected from the following: personal digital assistant (PDA, handheld device with computer, phone/fax and Internet), wireless application protocol (WAP) phone.

0113 Preferably, method 401, wherein the steps transforming (410), building (421) and generating (431) are performed for a plurality of development objects substantially simultaneously in a pipeline mode.

0114 Preferably, method 401, further comprising step storing the abstract syntax tree (241) in a tree library. It is an advantage; reuse of once provided abstract syntax trees.

0115 Preferably, method 401, further comprising interacting with a processing function to modify any of development object (DO, 105), intermediate object (IO, 215), template (151) and abstract syntax tree.

0116 Preferably, method 401, is performed by a builder (221), wherein step generating (431) is performed by a generator (231), and wherein the builder (221) and the generator (231) are provided (to the customer) in combination.

0117 FIG. 5 illustrates a diagram with software components to implement method 401 of the present invention: repository 101 (as input), transformer 210, intermediate objects 215, builder (B) 221, abstract syntax tree (AST) 241, templates 151 (arrow symbol), generator 231, runtime objects (RO 305 in form of source code (SC) 301. Method steps 410, 421, 431 are shown below. Manager 260 coordinates the operation.

0118 FIG. 6 illustrates details for step transforming: IO-UML-document 274,
0119 IO-XML-document 275, IO-XSLT processor 276, IO-XSL-stylesheet 277.

0120 FIG. 7 illustrates further details for step transforming by comparing a first model of the development object 105 and a second model of an intermediate object 205. By way of simplified example, the development object is designed to for a user interface (UI) element.

0121 Both models are slightly different. The second model is adapted to the source code language the runtime object (205, to be generated) . In the example, the runtime object 205 will be code in VBA (Visual Basic for Business Applications). VBA does not support inheritance. Therefore, the transformation (step 410) maps inheritance (in the first model) to sub- and super-associations (in the second model). This ensures that the properties and relations of the first model are properly transformed to the second model.

0122 Both models are illustrated by Unified Modeling Language. Symbols like arrows and diamonds are well known in the art. The underscore symbol is introduced in enhance readability.

Inheritance

0123 In the first model (upper half of the figure), for example, ARS stands for "Application Repository Services"; AROM stands for "Application Repository Object Model"; UI_Interaction_Comp stands for "Interaction Component, UI Layer". In the second model (lower half of the figure), IOM stands for

"Intermediate Object Model".

0124 FIG. 8 illustrates details for step building 421 by illustrations of template 241 and builder 221 with AST-class-provider 290, generation template 293,293-n, template-to-XML-converter 294, AST-XML-template 295, AST-XSLT-processor 296, AST-XSL-stylesheet 297, AST-node-class 298.

0125 FIG. 9 illustrate details for step building by showing the abstract syntax tree as a diagram in UML for an exemplary user interface. The class diagram is given in unified modeling language UML. The rectangles are divided into upper compartments (class name) and lower compartments (class attribute).

0126 FIG. 10 illustrates further details for step building by showing the names and properties for each node in the diagram of FIG. 9;

0127 Conveniently, such as class diagram is shown to the developer (function 1000, cf FIG.1) during design time.

0128 The names of the classes correspond to the names of the classes in the source code (that is generated).

0129 Node 0 has name "Node" and stands for generate (i.e. performing generating step)

0130 Node 1 has name "ApplClass" and has attributes that are input parameters "AppName" to indicate the name of the application, "GenVersion" to indicate the version of the application, "GenDate" to indicate the calendar date when the source code is generated (i.e. step building of method 401), "FreeCode" to add design-time model-specific source code sections, and "AppEventHandled" to determine whether the object of a basis class raises events or not (used below in the declaration of the MCore data member).

0131 Node 2 is a subnode to node 1 and stands for "PopupEnums", a single aggregation implemented as direct reference. Node 5 is a subnode to node 1 and stands for "BCDeclaration" having attributes "ContainerName" and

"BCName". Details for node 5 is also explained in reference to FIG. 13 (code lines framed). Nodes 4, 5, 6, 7, 8, 9, 10, 11, 14, 15, 16 are multiple aggregations (cf. asterisk symbol) and list variables.

0132 FIG. 11 illustrates details for step building by showing the abstract syntax tree of FIGS. 9-10 as a file with code segments for each class. It can be used to generate source code in different languages.

0133 For convenience of explanation, columns within in dashed vertical lines indicate line numbers and node numbers (cf. FIGS. 9-10)

0134 for convenience of explanation, the figure notes grammar elements with bold italics.

0135 Code classes (also called "production rules") are numbered from 1 to 16 (corresponding to the nodes). The production rules have placeholders (e.g., indicated by "&" symbols). Generating replaces the placeholders by actual values, depending on input data (i.e. concrete application name replaces the placeholder &AppName&) resulting in source code 301.

0136 Using <tagname> for the name of a class (e.g., class "ApplClass" for node 1), start tags and end tags use this name to identify code for the class

0137 The following convention applies: $<tagname> for the start tag; $End<tagname> for the end tag; further asterisk * and plus + signs follow the $ sign; asterisk * for a number of occurrences between 0 (no occurrence) and N; plus + for a number of occurrences between 1 (exactly one occurence) and N.

0138 Alternatives are indicated by vertical stroke (cf. Backus Naur). For example, the class "EventHandler" for node 11 can result in slightly different code for the parameters. The "EventHandler" has 3 parameters. First parameter ("Signature") and second parameter ("Code") go into source code mandatory; the third parameter goes into the source either as "Normal" or as "MoreThanOneForSameEvent".

0139 As an example, node 5 "$*BCDeclaration(ContainerName, BCName): Private C&ContainerName& as C&BCName& EndBCDeclaration" is replaced by place holder.

0140 FIG. 12 illustrates a classification of templates and corresponding abstract syntax trees, by examples from "application class" to "business object class".

0141 FIG. 13 illustrates a division of templates into template files by example for the "application class" template, with the assumption one template file to have one production rule.

0142 FIG. 14 illustrates an example template file with an example production rule "BCDeclaration".

0143 FIG. 15 illustrates providing an abstract syntax tree library.

0144 FIG. 16 illustrates an example of a generator, with computer instructions perform step generating (of the method of FIG. 4) from the abstract syntax tree (AST), thereby instantiating the AST with data.

0145 FIG. 17 illustrates exemplary data, used during step generating.

0146 FIG. 18 illustrates an example for an application class file being a runtime object, obtained in step generating.

0147 FIG. 19 illustrates an example for a generation template, used to provide a generator (for step generating, cf. FIG. 16).

0148 FIG. 20 illustrates an example of an AST-XML-template 295-08 (result of converting 4XX generation template 293-08 to AST-XML-template 295-08 by using template-to-XML-converter 294) used for providing the generator.

0149 FIG. 21 illustrates an example of an XSL stylesheet used to provide the generator.

0150 FIG. 22 illustrates a simplified diagram of a computer network system. It illustrates a simplified block diagram of exemplary computer system 999 having a plurality of computers 900, 901, 902 (or even more, cf. FIG. 1 first, second, third computer).

0151 Computer 900 can communicate with computers 901 and 902 over network 990. Computer 900 has processor 910, memory 920, bus 930, and, optionally, input device 940 and output device 950 (I/O devices, user interface 960). As illustrated, the invention is implemented by computer program product 100 (CPP), carrier 970 and signal 980.

0152 In respect to computer 900, computer 901/902 is sometimes referred to as "remote computer", computer 901/902 is, for example, a server, a peer device or other common network node, and typically has many or all of the elements described relative to computer 900.

0153 Computer 900 is, for example, a conventional personal computer (PC), a desktop device or a hand-held device, a multiprocessor computer, a pen computer, a microprocessor-based or programmable consumer electronics device, a minicomputer, a mainframe computer, a personal mobile computing device, a mobile phone, a portable or stationary personal computer, a palmtop computer or the like.

0154 Processor 910 is, for example, a central processing unit (CPU), a micro-controller unit (MCU), digital signal processor (DSP), or the like.

0155 Memory 920 is elements that temporarily or permanently store data and instructions. Although memory 920 is illustrated as part of computer 900, memory can also be implemented in network 990, in computers 901/902 and in processor 910 itself (e.g., cache, register), or elsewhere. Memory 920 can be a read only memory (ROM), a random access memory (RAM), or a memory with other access options. Memory 920 is physically implemented by computer-readable media, for example: (a) magnetic media, like a hard disk, a floppy disk, or other magnetic disk, a tape, a cassette tape; (b) optical media, like optical disk (CD-ROM, digital versatile disk - DVD); (c) semiconductor media, like DRAM, SRAM, EPROM, EEPROM, memory stick.

0156 Optionally, memory 920 is distributed. Portions of memory 920 can be removable or non-removable. For reading from media and for writing in media, computer 900 uses well-known devices, for example, disk drives, or tape drives.

0157 Memory 920 stores modules such as, for example, a basic input output system (BIOS), an operating system (OS), a program library, a compiler, an interpreter, and a text- processing tool. Modules are commercially available and can be installed on computer 900. For simplicity, these modules are not illustrated.

0158 CPP 100 has program instructions and - optionally - data that cause processor 910 to execute method steps of the present invention. In other words, CPP 100 can control the operation of computer 900 and its interaction in network system 999 so that is operates to perform in accordance with the invention. For example and without the intention to be limiting, CPP 100 can be available as source code in any programming language, and as object code ("binary code") in a compiled form.

0159 Although CPP 100 is illustrated as being stored in memory 920, CPP 100 can be located elsewhere. CPP 100 can also be embodied in carrier 970.

0160 Carrier 970 is illustrated outside computer 900. For communicating CPP 100 to computer 900, carrier 970 is conveniently inserted into input device 940. Carrier 970 is implemented as any computer readable medium, such as a medium largely explained above (cf. memory 920). Generally, carrier 970 is an article of manufacture having a computer readable medium with computer readable program code to cause the computer to perform methods of the present invention. Further, signal 980 can also embody computer program product 100.

0161 Having described CPP 100, carrier 970, and signal 980 in connection with computer 900 is convenient. Optionally, further carriers and further signals embody computer program products (CPP) to be executed by further processors in computers 901 and 902.

0162 Input device 940 provides data and instructions for processing by computer 900. Device 940 can be a keyboard, a pointing device (e.g., mouse, trackball, cursor direction keys), microphone, joystick, game pad, scanner, or disc drive. Although the examples are devices with human interaction, device 940 can also be a device without human interaction, for example, a wireless receiver (e.g., with satellite dish or terrestrial antenna), a sensor (e.g., a thermometer), a counter (e.g., a goods counter in a factory). Input device 940 can serve to read carrier 970.

0163 Output device 950 presents instructions and data that have been processed. For example, this can be a monitor or a display, (cathode ray tube (CRT), flat panel display, liquid crystal display (LCD), speaker, printer, plotter, vibration alert device. Output device 950 can communicate with the user, but it can also communicate with further computers.

0164 Input device 940 and output device 950 can be combined to a single device. Any device 940 and 950 can be provided optional.

0165 Bus 930 and network 990 provide logical and physical connections by conveying instruction and data signals. While connections inside computer 900 are conveniently referred to as "bus 930", connections between computers 900-902 are referred to as "network 990". Optionally, network 990 includes gateways which are computers that specialize in data transmission and protocol conversion.

0166 Devices 940 and 950 are coupled to computer 900 by bus 930 (as illustrated) or by network 990 (optional). While the signals inside computer

900 are mostly electrical signals, the signals in network are electrical, electromagnetic, optical or wireless (radio) signals.

0167 Networks are commonplace in offices, enterprise-wide computer networks, intranets and the Internet (e.g., world wide web). Network 990 can be a wired or a wireless network. To name a few network implementations, network 990 can be, for example, a local area network (LAN), a wide area network (WAN), a public switched telephone network (PSTN); a Integrated Services Digital Network (ISDN), an infra-red (IR) link, a radio link, like Universal Mobile Telecommunications System (UMTS), Global System for Mobile Communication (GSM), Code Division Multiple Access (CDMA), or satellite link.

0168 A variety of transmission protocols, data formats and conventions is known, for example, as transmission control protocol/internet protocol (TCP/IP), hypertext transfer protocol (HTTP), secure HTTP, wireless application protocol (WAP), unique resource locator (URL), a unique resource identifier (URI), hypertext markup language (HTML), extensible markup language (XML), extensible hypertext markup language (XHTML), wireless markup language (WML), Standard Generalized Markup Language (SGML).

0169 Interfaces coupled between the elements are also well known in the art. For simplicity, interfaces are not illustrated. An interface can be, for example, a serial port interface, a parallel port interface, a game port, a universal serial bus (USB) interface, an internal or external modem, a video adapter, or a sound card.

0170 Computer and program are closely related. As used, phrases, such as "the computer provides" and "the program provides", are convenient abbreviation to express actions by a computer that is controlled by a program.

0171 The present invention can also be considered as a process for providing computer source code (301) to a customer, wherein the process has a first sub-process performed by at least one manufacturer (at development time) and a second sub-process performed by the customer (at process time), wherein the sub-processes comprises step as follows: the first sub-process, providing a template (151) and a builder (221), transferring the template (151) and the builder (221) to the customer (C); in the second sub-process, providing a repository (101) with development objects (105) that implements customer specific data, performing method, thereby providing source code (that implements customer specific data).

0172 The source code (301) is subsequently compiled to an application.

0173 Further, a method for providing source code (301) (i.e. runtime code) for subsequent execution in a runtime framework (310), the method (400) comprising the following steps: providing first code that enables a computer to transform (410) a development object (105) into an intermediate object model (215); providing second code that enables the computer to build (421) an abstract syntax tree (241) from the intermediate object model (215) by using a template; and providing third code that enables the computer to generate (431) the source code from the abstract syntax tree (241) by linking the abstract syntax tree (241) while preserving the structure of the template.

Reference numbers

1, 2 ...16	nodes in AST, classes in template
1000	developer function
101	development object repository
105	development objects (DO)
106	application development environment
1xx	relating to design time

2000	processing function
215	intermediate object model (IOM)
221	builder
241	abstract syntax tree (AST)
249	syntax library
260	process manager
274	IO-UML-document
275	IO-XML-document
276	IO-XSLT processor
277	IO-XSL-stylesheet
279	Intermediate object (IO) library
290	AST-class-provider
293, 293-n	generation template
294	template-to-XML-converter
295	AST-XML-template
296	AST-XSLT-processor
297	AST-XSL-stylesheet
298	AST-node-class
2xx	relating to transition time
3000	user function
301	source code
305	runtime objects
310	runtime framework
3xx	(relating to run time)
401	method
410	transforming
421	building
431	generating

Acronyms

AST	abstract syntax tree
B	builder
CF	code fragments
CPP	computer program product
DEV	developer function
DO	development object
DOM	Document Object Model
G	generator
HTML	Hypertext Markup Language
IOM	Intermediate Object Model
k	index for classes
n	index for nodes
OO	Object Oriented
PDA	Personal Digital Assistant
PM	process manager
PR	production rule
PRO	processing function
SC	source code
T	template
TRA	transformer
UML	Unified Modeling Language
URL	Uniform Resource Locator
VBA	Visual Basic for Applications
XML	Extensible Markup Language
XSL	Extensible Style Language
XSLT	Extensible Style Language Transformation

Claims

1. Method (401) for use in a computer for processing a development object (DO, 105) into a runtime object (305), the method (400) comprising the following steps:
transforming (410) a development object (DO, 105) into an intermediate object (IO, 215);
building (421) an abstract syntax tree (AST, 241) from the intermediate object (215) by using a template (151); and
generating (431) the runtime object (205) from the abstract syntax tree (241) while preserving the structure of the template (151).

2. The method (401) of claim 1, wherein the development object (DO, 105) comprises meta-data for an application with information about the business logic of the application.

3. The method (401) of claim 1, wherein the development object (DO, 105) has been provided in a visual environment by drag and drop declarations.

4. The method (401) of claim 1, wherein the runtime object (305) is an object in source code (301).

5. The method (401) of claim 1, wherein step transforming (410) involves a development object (DO) of a business application.

6. The method (401) of claim 1, wherein in step building (421), the template (151) uses language elements suitable for files selected from the group of: application class file, application project file, common registry file, machine specific registry file, business component class file, tileset class file, tile HTML file, and business object class file.

7. The method (401) of claim 1, wherein generating (431) into source code (301) comprises to generate runtime objects in languages selected from the group of Java, Visual Basic for Applications (VBA), Hyper Text Markup Language (HTML).

8. The method (401) of claim 1, wherein generating (431) the runtime object comprises to replace placeholders in the abstract syntax tree (241) with data.

9. The method (401) of claim 1, wherein transforming (410) comprises to receive the development object (105) from a repository (101).

10. The method (401) of claim 1, wherein transforming (410) is performed for a plurality of development objects (105).

11. The method (401) of claim 9, wherein transforming (410) the plurality of development objects (105) comprises to preserve the relations between the development objects (105).

12. The method (401) of claim 9, wherein transforming (410) comprises to use development objects (105) based on a first model and to provide intermediate objects (215) based on a second model, wherein the first model and the second model have the same meta-model.

13. The method (401) of claim 12, wherein transforming (410) comprises to keep the properties and relations in the first model and in the second model.

14. A computer program product, which can be loaded into an internal memory of a digital data processing means and has computer program code means, which carry out the method of any of claims 1-13 when they are loaded and run on one or more data processing means

15. Computer system for performing the method of any of claims 1-13.

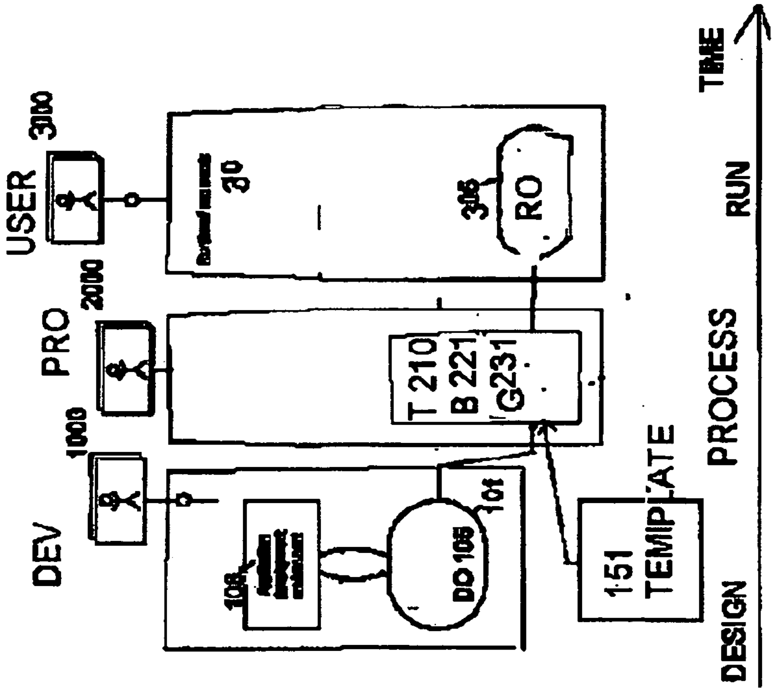
DEV
1000
108
DO 105
101
PRO
2000
T 210
B 221
G231
USER
3000
310
305
RO
151
TEMPLATE
DESIGN
PROCESS
RUN
TIME

FIG. 1

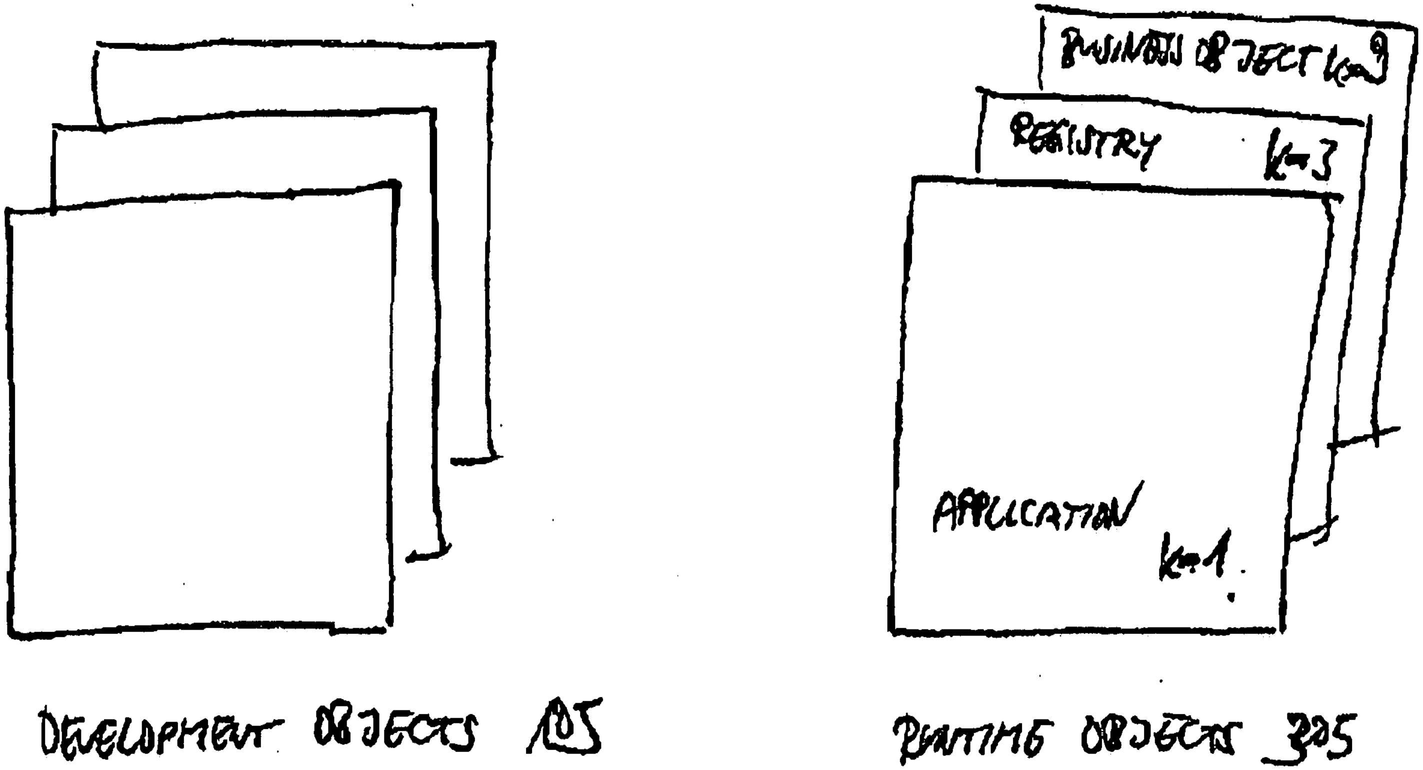
BUSINESS OBJECT k=9
REGISTRY k=3
APPLICATION k=1.
DEVELOPMENT OBJECTS 105
RUNTIME OBJECTS 305

FIG. 2

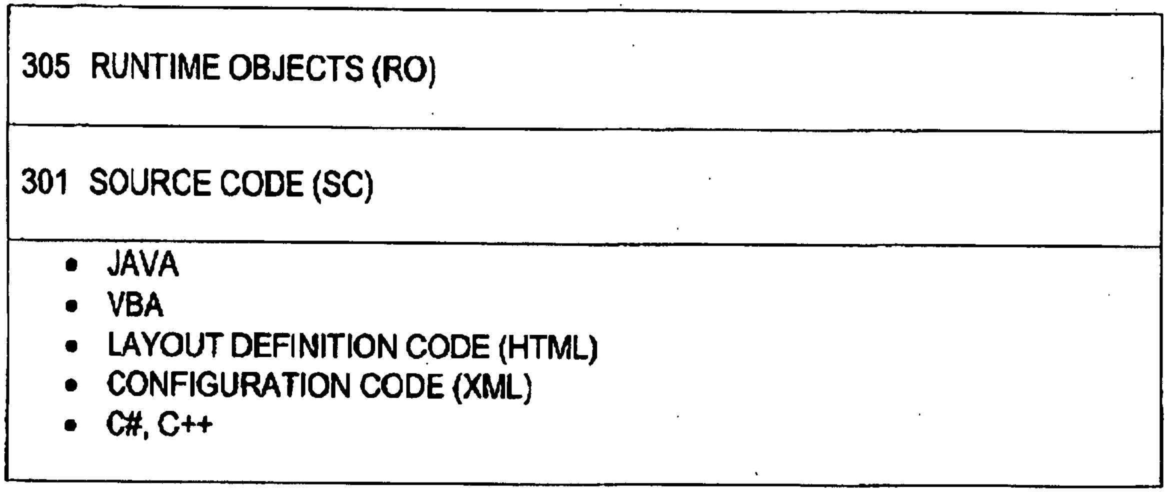
305 RUNTIME OBJECTS (RO)
301 SOURCE CODE (SC)
• JAVA
• VBA
• LAYOUT DEFINITION CODE (HTML)
• CONFIGURATION CODE (XML)
• C#, C++

FIG. 3

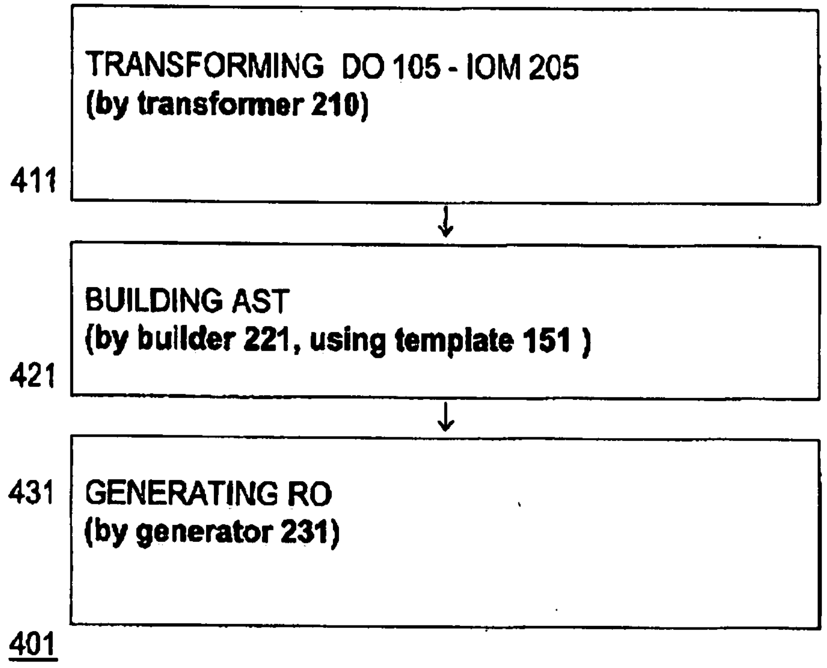
TRANSFORMING DO 105 - IOM 205
(by transformer 210)
411
BUILDING AST
(by builder 221, using template 151)
421
GENERATING RO
(by generator 231)
431
401

FIG. 4

2003P00111EP

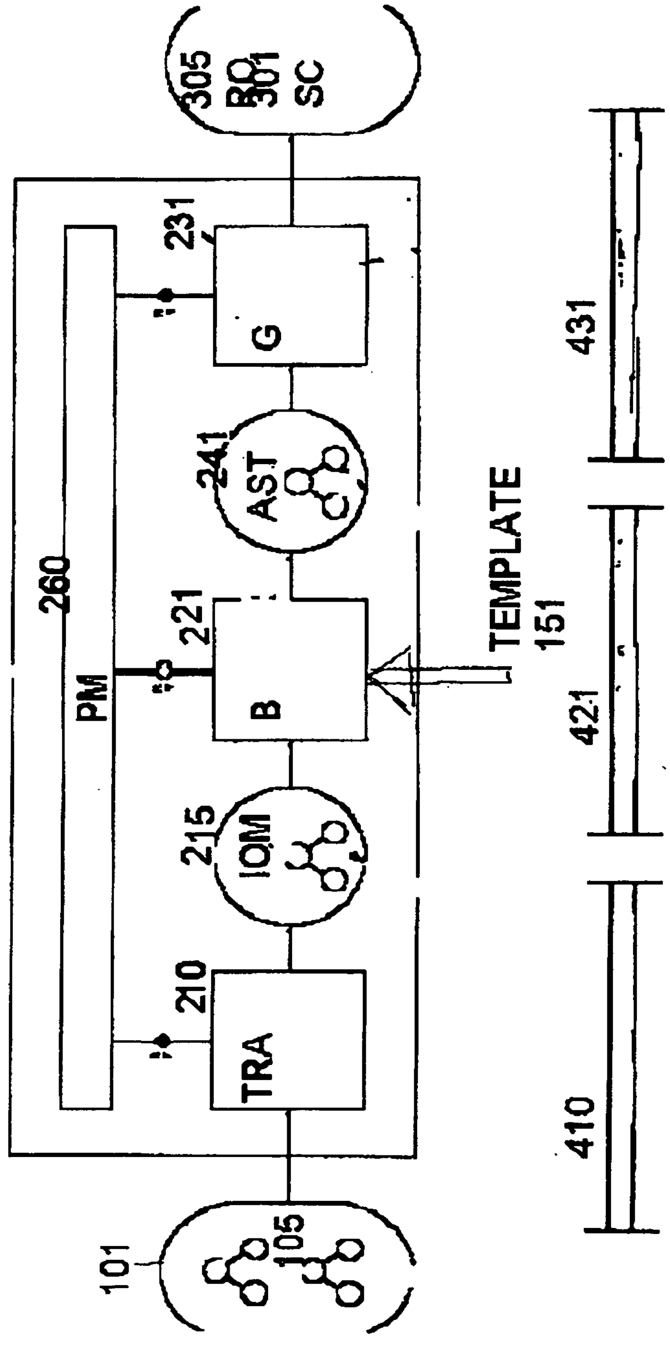

FIG. 5

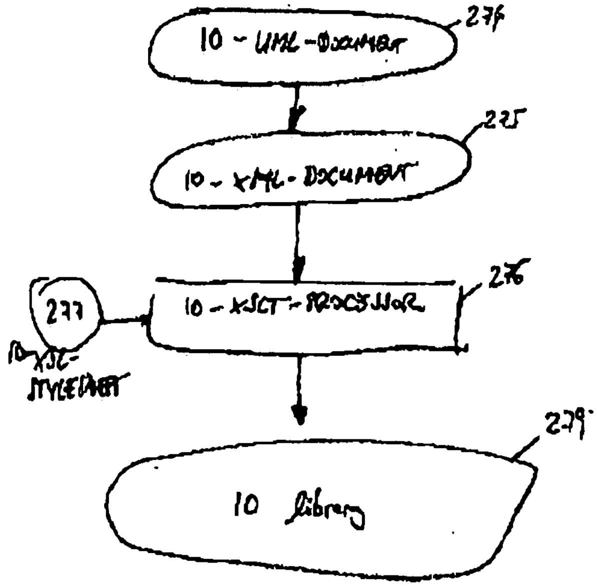
10 - UML-DOCUMENT
274
10 - XML-DOCUMENT
275
10 - XSLT - PROCESSOR
276
277
IO-XSL-STYLESHEET
10 library
279

FIG. 6

2003P00111EP

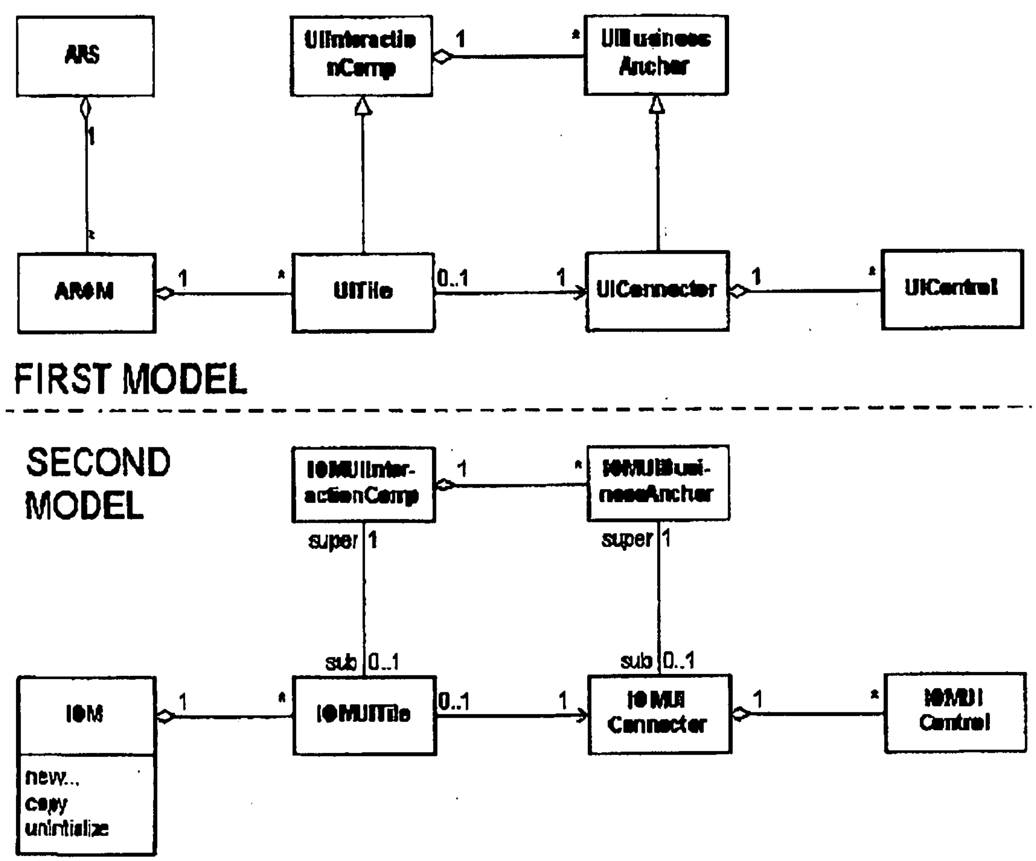

FIG. 7

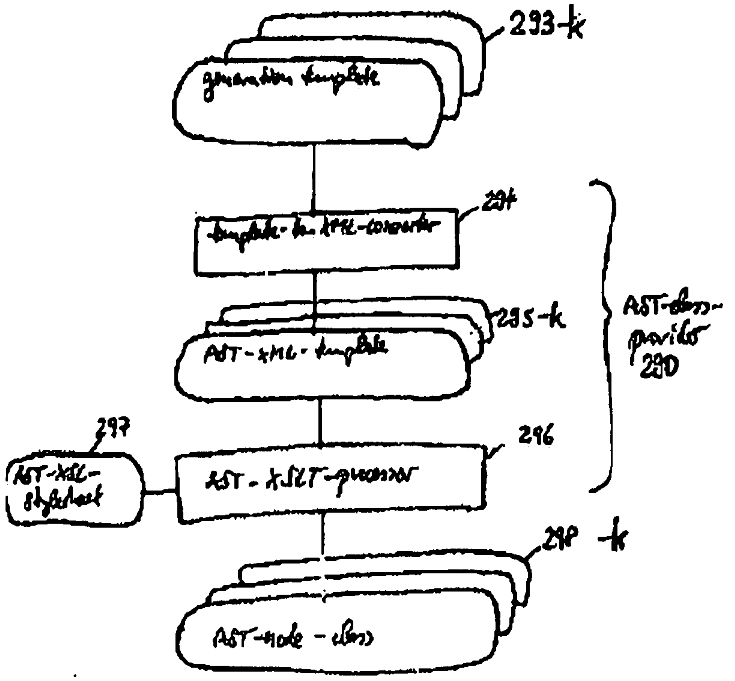
generation template
293-k
template-to-XML-converter
294
AST-XML-template
295-k
AST-class-provider
290
AST-XSL-stylesheet
297
AST-XSLT-processor
296
AST-node-class
298-k

FIG. 8

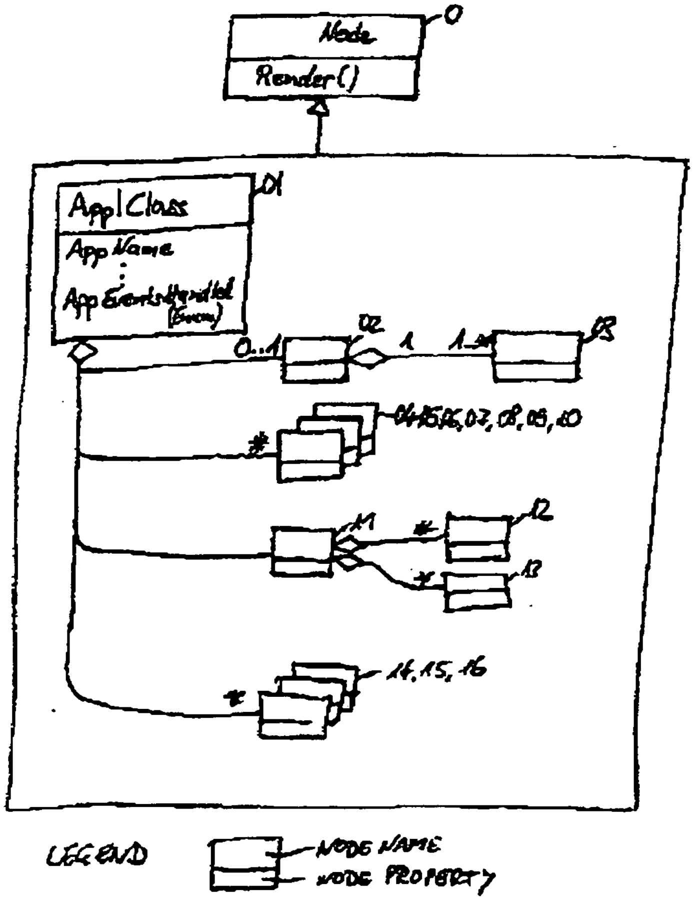
Node
Render()
0
ApplClass
AppName
:
AppEventHandler
(Event)
01
0..1
02
1
1..n
03
*
04,05,06,07,08,09,10
11
*
12
*
13
*
14,15,16
LEGEND
NODE NAME
NODE PROPERTY

FIG. 9

node/class index n	name of node/class	properties
0		render()
01	ApplClass	ApplName GenVersion GenDate FreeCode AppEventsHandled (Enum)
02	PopupEnums	
03	PopupEnumValue	PopupName Index
04	EventDeclaration	EventSignature
05	BCDeclaration	ContainerName BCName
06	AnchorDeclaration	case(enum) AnchorName
07	CustPropertyDecl	Visibility (enum) Name Type
08	AnchorAssign	AnchorName Index
09	ComponentAssign	BCName Index
10	SupplyFunctionCall	AnchorName
11	EventHandler	case (enum) Signature Code
12	DispatchCall	CallStatement
13	IndexedEventHandler	Signature Code
14	CustomMethod	Signature Code SubOrFunction(Enum)
15	RaiseEventMethod	NormalizedSignature Modified Signature
16	SupplyFunction	AnchorName Code

FIG. 10

line	Node	
01	1	***$ApplClass(AppName, GenVersion, GenDate, FreeCode,***
02	1	***AppEventsHandled(yes:***"WithEvents ", ***no:"")):***
03	1	VERSION 1.0 CLASS
04	1	BEGIN
05	1	MultiUse = -1 'True
06	1	END
07	1	Attribute VB_Name = "A***&AppName&***"
08	1	Attribute VB_GlobalNameSpace = False
09	1	Attribute VB_Creatable = True
010	1	Attribute VB_PredeclaredId = False
011	1	Attribute VB_Exposed = False
012	1	' START CODE Application Class File
013	1	' Generated by UI-Generator ***&GenVersion&*** on ***&GenDate&***
014	1	Option Explicit
015	1	Implements ICustApplication
016	1	Implements ICustIAC
017	1 2	***$PopupEnums():***
018	1 2	private Enum enm_PopupTilesets
019	1 2 3	***$+PopupEnumValue(PopupName, Index):***
020	1 2 3	T***&PopupName&*** = ***&Index&***
021	1 2 3	***EndPopupEnumValue***
022	1	End Enum
023	1 2	***EndPopupEnums***
024	1 4	***$*EventDeclaration(eventSignature):***
025	1 4	***&EventSignature&***
026	1 4	***EndEventDeclaration***
027	1	Private &AppEventsHandled&mCore As CoreApplication
028	1 5	***$*BCDeclaration(ContainerName, BCName):***
029	1 5	Private C***&ContainerName&*** as C***&BCName&***
030	1 5	***EndBCDeclaration***
031	1 6	***$*AnchorDeclaration(AnchorName,***
032	1 6	***alternatives: NoEventsHandled*** | ***Eventshandled):***
033	1 6	private a***&AnchorName&*** as CoreBusinessAnchor |
034	1 6	private withEvents a***&AnchorName&*** as CoreBusinessAnchor
035	1 6	***EndAnchorDeclaration***
036	1 7	***$*CustPropertyDecl(Visiblity(public:***"Public", ***private:***"Private"***), Name, Type):***
037	1 7	***&visibility&*** p***&Name&*** as ***&Type&***

FIG: 11 A

```
038  1 7   $End$CustPropertyDecl
039  1        &FreeCode&
040  1     Private Sub ICustIAC_assign( _
041  1     objects() As Variant, ByVal level As
           EAssignLevel)
042  1     Select Case level
043  1             Case ealCore
044  1                     Set mCore = objects(0)
045  1                     Set gServices = objects(1)
046  1                     'mDebugMonitor = objects(2)
047  1                     Set gFactory = objects(3)
048  1                     Set gApplication = objects(4)
049  1                     Set gBOLSettings = objects(5)
050  1                 Case ealAnchors
051  1 8   $*AnchorAssign(AnchorName, index):
052  1 8               set a&AnchorName& =
           objects(&index&)
053  1 8   $End$AnchorAssign
054  1                     Case ealIACs
055  1 9   $*ComponentAssign(BCName, index):
056  1 9               set c&BCName& = objects(&index&)
057  1 9   $End$ComponentAssign
058  1     End Select
059  1     End Sub
060  1     Private Function
           ICustIAC_callSupplyFunction(_
061  1     ByVal supplyName As String, _
062  1     ByVal parentAnchor As CoreBusinessAnchor, _
063  1     content As Object, _
064  1     pos As Long) As Boolean
065  1     On Error GoTo ExitFct
066  1     Select Case supplyName
067  1 10  $*SupplyFunctionCall(AnchorName):
068  1 10          case a&AnchorName&_onSolve
069  1 10              a&AnchorName&_onSolve
           parentAnchor, content, pos
070  1 10  $End$SupplyFunctionCall
071  1             End Select
072  1             ICustIAC_callSupplyFunction = True
073  1             Exit Function
074  1     ExitFct:
075  1             ICustIAC_callSupplyFunction = False
```

FIG. 11 B

```
1          End Function
1 11       $*EventHandler(Signature, Code,
1 11       alternatives: Normal |
           MoreThanOneForSameEvent):
1 11          Private Sub &Signature&
1 11             &Code&
1 11          End Sub
1 11          |
1 11          Private Sub &Signature&
1 11 12       $+DispatchCall(CallStatement):
1 11 12          &CallStatement&
1 11 12       $End$DispatchCall
1 11          End Sub
1 11 13       $+IndexedEventHandler(Signature, Code):
1 11 13       Private Sub &Signature&
1 11 13          &Code&
1 11 13       End Sub
1 11 13       $End$IndexedEventHandler
1 11       $End$EventHandler
1 14       $*CustomMethod(Signature, Code,
           SubOrFunction("Sub", "Function"):
1 14          &Signature%
1 14             &Code&
1 14          End &SubOrFunction&
1 14       $End$CustomMethod
1 15       $*RaiseEventMethod(NormalizedSignature,
           ModifiedSignature):
1 15          private sub
           RaiseEvent__&NormalizedSignature&
1 15             RaiseEvent &ModifiedSignature&
1 15          End Sub
1 15       $End$RaiseEventMethod
1 16       $*SupplyFunction(AnchorName, code):
1 16          Public Sub a&AnchorName&__onSolve( _
1 16       parentAnchor as UFCore.CoreBusinessAnchor, _
1 16       destination as Object, _
1 16       pos as Long)
1 16             &Code&
1 16          End Sub
1 16       $End$SupplyFunction
1          $End$AppClass
```

FIG. 11 C

types of generation template 293 (distinguished by grammar)	types of AST abstract syntax tree library type	types of RO
293-k, with k class index		
application class template 293-1	application class AST	application class file
application project template 293-2	application project AST	application project file
common registry template 293-3	common registry AST	common registry file
machine specific registry template 293-4	machine specific registry AST	machine specific registry file
business component class template 293-5	business component class AST	business component class file
tileset class template 293-6	tileset class AST	tileset class file
tile class template293-7	tile class AST	tile class file
tile html template 293-8	tile html AST	tile html file
business object class template 293-9	business object class AST	business object class file

FIG. 12

generation
template 293
application
class
template
293-1*

template file	with production rules (PR):	AST library - a single file with nodes node/class index n
293-1-01	"ApplClass"	01
293-1-02	"PopupEnums"	02
293-1-03	"PopupEnumValue"	03
293-1-04	"EventDeclaration"	04
293-1-05	"BCDeclaration"	05
293-1-06	"AnchorDeclaration"	06
293-1-07	"CustPropertyDecl"	07
293-1-08	"AnchorAssign"	08
293-1-09	"ComponentAssign"	09
293-1-10	"SupplyFunctionCall"	10
293-1-11	"EventHandler"	11
293-1-12	"DispatchCall"	12
293-1-13	"IndexedEventHandler"	13
293-1-14	"CustomMethod"	14
293-1-15	"RaiseEventMethod"	15
293-1-16	"SupplyFunction"	16

application
project
293-2*
...

application
project
293-8*

* being template projects

FIG. 13

```
$*BCDeclaration(ContainerName, BCName):

Private C&ContainerName& as C&BCName&

$End$BCDeclaration
```

& & placeholders for "ContainerName" and for "BCName"

FIG. 14

2003P00111EP

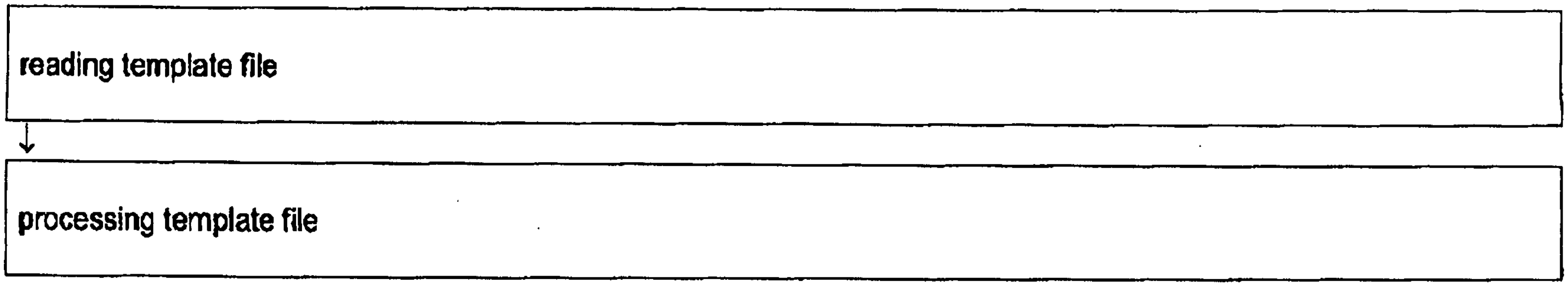

FIG. 15

```
public string Render()
     {    StringBuilder code;
          List.Iterator i;
          string innerCode;
          code = new StringBuilder(Template);
          code.Replace("&AppName&", AppName);
          code.Replace("&GenVersion&", GenVersion);
          code.Replace("&GenDate&", GenDate);
          code.Replace("&FreeCode&", FreeCode);
          switch (AppEventsHandled)
          {    case AppClass_AppEventsHandled.Yes:
                    code.Replace("&AppEventsHandled&", "WithEvents ");
                    break;
               case AppClass_AppEventsHandled.No:
                    code.Replace("&AppEventsHandled&", "");
                    break;     }
```

starting instructions

For node 01, replacing parameters
with values for each property
for property "AppName"
for property "GenVersion"
for property "FreeCode"
for property "ApplEventsHandled"
(with case distinction)

FIG: 16 A

```
        innerCode = "";
        if (mPopupEnums != null)
        { innerCode = mPopupEnums.Render();
        }
        code.Replace("&PopupEnums&", innerCode);
        innerCode = "";
        for (i = mEventDeclarations.Begin;
        i != mEventDeclarations.End; i++)
        {     EventDeclaration lEventDeclaration;
              lEventDeclaration = (EventDeclaration) i.Value;
              innerCode = innerCode + lEventDeclaration.Render(); }
              code.Replace("&*EventDeclaration&", innerCode);
    ...

              return code.ToString(); }
```

for node 02 "PopupEnums", being a sub-node to node 01, replacing parameters

for node 04 "EventDeclarations"; being a further sub-node to node 01; replacing parameters

replacing for nodes 05 to 16 that are subnodes to node 01 closing instructions

FIG. 16 B

2003P00111EP

AppName	ALPHA
GenVersion	2
GenDate	31 December 2002
FreeCode	///
AppEventsHandled	Yes
...	

FIG. 17

```
line  Node
01    1
02    1
03    1         VERSION 1.0 CLASS
04    1         BEGIN
05    1            MultiUse = -1          'True
06    1         END
07    1         Attribute VB_Name = "AALPHA"
08    1         Attribute VB_GlobalNameSpace = False
09    1         Attribute VB_Creatable = True
010   1         Attribute VB_PredeclaredId = False
011   1         Attribute VB_Exposed = False
012   1        ' START CODE Application Class File
013   1        ' Generated by UI-Generator 2 on 31 December
               2002
014   1        Option Explicit
015   1        Implements ICustApplication
016   1        Implements ICustIAC
017   1 2      $PopupEnums():
018   1 2         private Enum enm_PopupTilesets
019   1 2 3    $+PopupEnumValue(PopupName, Index):
020   1 2 3    T&PopupName& = &Index&
021   1 2 3    $End$PopupEnumValue
022   1        End Enum
023   1 2      $End$PopupEnums
024   1 4      $*EventDeclaration(eventSignature):
025   1 4      &EventSignature&
026   1 4      $End$EventDeclaration
027   1           Private &AppEventsHandled&mCore As
               CoreApplication
028   1 5      $*BCDeclaration(ContainerName, BCName):
029   1 5      Private C&ContainerName& as C&BCName&
030   1 5      $End$BCDeclaration
```

FIG. 18

1	$HRMLPage (Name, Color) <HTML> <HEAD> <TITLE>&Name&</TITLE> </HEAD>	start of production rule for the content of a page, with parameters Name and Color
1	<BODY BGCOLOR=&Color&>	
1 2	$Table(Caption): &Caption&	start of production rule for the content of a table
1 2	<TABLE>	
1 2 3	$+TableRow(): <TR> <TD>Hello</TD> </TR> EndTableRow	production rule for the content of a table-row this production rule is subordinated to rule for the content of the table
1 2 3	</TABLE>	
1 2 3	EndTable	end of production rule for the content of the table
1	</BODY> </HTML>	
1	EndHTMLPage	end of production rule for the content of the page

generation template 293-08

FIG. 19

	<?xml version="1.0" ?>
	    <library>
	    <production name="HTMLPage">
	    <param name="Name" />
	    <param name="Color" />
		
		indicating the start of a library
		indicating production name (the name of the main node?)
		introducting parameters

	<template> <![CDATA[<HTML> <HEAD> <TITLE><ph>Name</ph></TITLE> </HEAD> <BODY BGCOLOR=<ph>Color</ph>> <pph>Table</pph> </BODY> </HTML>]]> </template>	start of template taking care about parameters Name and Color by definition, no XML-code allowed in CDATA-section end of template

FIG. 20A

	<production name="Table" Cardinality="single"> <param name="Caption" />	
	<template> <![CDATA[<ph>Caption</ph> <TABLE> <pph>TableRow</pph> </TABLE>]]> </template> <production name="TableRow" Cardinality="multiple"> <template> <![CDATA[<TR> <TD>Hello</TD> </TR>]]> </template> </production> </production> </production> </library>	

FIG: 20 B

```
<xsl: stylesheet ... >
...
color = green
...
</xsl stylesheet>
```

AST-XSL-stylesheet 297

FIG. 21

2003P00111EP

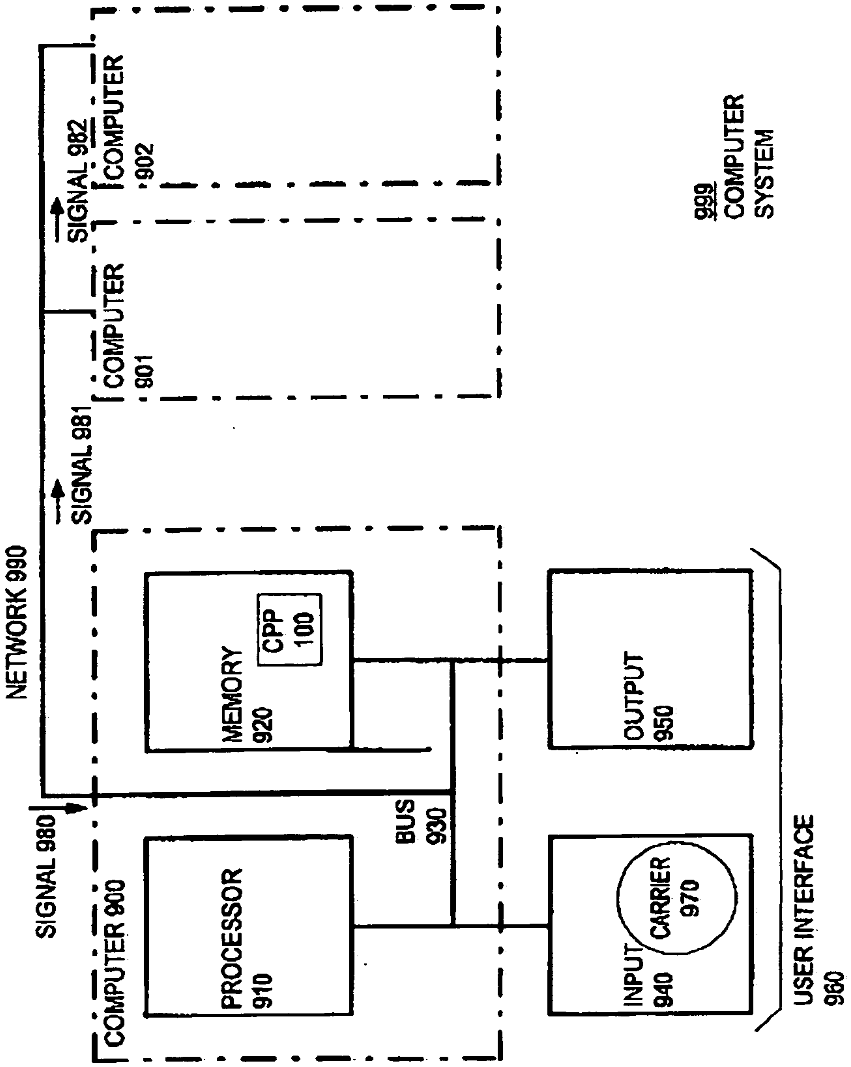

FIG. 22